

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Замыкания в языке Python»**

**Отчет по лабораторной работе № 2.11
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2022

Цель работы: приобретение навыков по работе с замыканиями при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * **Repository name ***

Yana-Kh ▾ / Lab-14-OPJ ✓

Great repository names are short and memorable. Need inspiration? How about **laughing-potato?**

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Python ▾

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License ▾

This will set **main** as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/Lab-14-OPJ.git
Cloning into 'Lab-14-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
C:\Users\ynakh\OneDrive\Рабочий стол\Git>_
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-14-OPJ>git add .
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-14-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-14-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-14-OPJ>git branch
* develop
  main
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

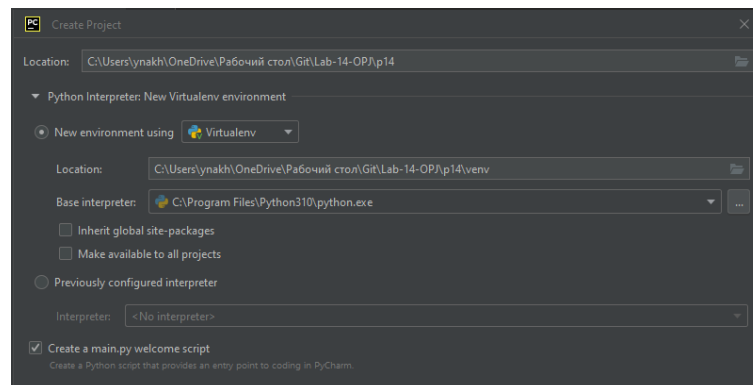


Рисунок 5 – Создание проекта PyCharm в папке репозитория

7. Проработайте пример лабораторной работы.

Код:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def fun1(a):
    x = a * 3

    def fun2(b):
        nonlocal x
        return b + x

    return fun2

if __name__ == '__main__':
    test_fun = fun1(4)
    print("ex1")
    print(test_fun(7))

    print("\nex2")
    tpl = lambda d, e: (d, e)
    s = tpl(1, 2)
    print(s)
    f = tpl(3, s)
    print(f)
    c = tpl(s, f)
    print(c)
```

```
ex1
19

ex2
(1, 2)
(3, (1, 2))
((1, 2), (3, (1, 2)))
```

Рисунок 6 – Результат работы программы

8. Решите индивидуальное задание согласно своему варианту.

Вариант 32(2). Используя замыкания функций, объявите внутреннюю функцию, которая заключает строку *s* (*s* – строка, параметр внутренней функции) в произвольный тег, содержащийся в переменной *tag* – параметре внешней функции. Далее, на вход программы поступает две строки: первая с тегом, вторая с некоторым содержимым. Вторую строку нужно поместить в

тег из первой строки с помощью реализованного замыкания. Результат выведите на экран.

Код:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def get_tag(tag):
    x = tag

    def get_str(s):
        nonlocal x
        return ": ".join([x, s])
    return get_str

if __name__ == '__main__':
    test_fun = get_tag("test tag")
    print(test_fun("hello"))
```

```
test tag: hello

Process finished with exit code 0
```

Рисунок 9 – Результат работы программы

12. Зафиксируйте сделанные изменения в репозитории.

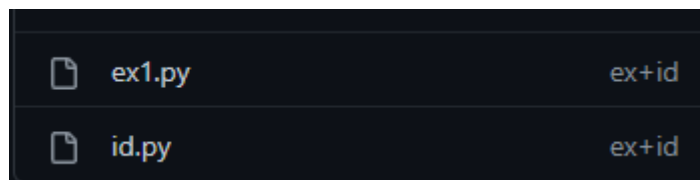


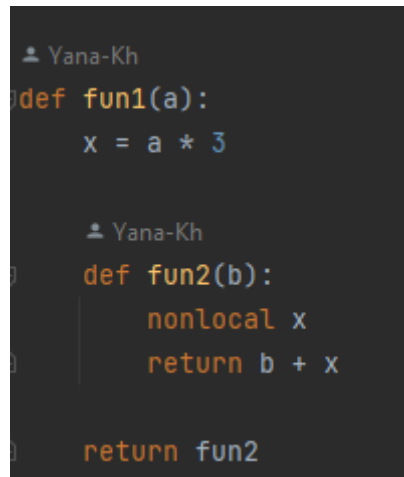
Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы

1. Что такое замыкание?

Замыкание – это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами.

2. Как реализованы замыкания в языке программирования Python?



```
def fun1(a):  
    x = a * 3  
  
    def fun2(b):  
        nonlocal x  
        return b + x  
  
    return fun2
```

3. Что подразумевает под собой область видимости Local?

Эту область видимости имеют переменные, которые создаются и используются внутри функций.

4. Что подразумевает под собой область видимости Enclosing?

Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для её вложенной функции находится в enclosing области видимости.

5. Что подразумевает под собой область видимости Global?

Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py)

6. Что подразумевает под собой область видимости Built-in?

Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости.

7. Как использовать замыкания в языке программирования Python?

В случае с реализацией выше:

```
test_fun = fun1(4)
print("ex1")
print(test_fun(7))
```

```
19
```

8. Как замыкания могут быть использованы для построения иерархических данных?

```
tpl = lambda d, e: (d, e)
s = tpl(1, 2)
print(s)
f = tpl(3, s)
print(f)
c = tpl(s, f)
print(c)
```

```
(1, 2)
(3, (1, 2))
((1, 2), (3, (1, 2)))
```